MITRE

CLINICAL QUALITY MEASURE LOGIC AND IMPLEMENTATION GUIDANCE

Implementing Computation of Specific Occurrences

March 22, 2013

This page intentionally left blank.

Table of Contents

This page intentionally left blank.

1 Purpose

Implementing specific occurrences is by far the most complex aspect of correctly implementing the electronic Clinical Quality Measures (eCQMs) included in the Electronic Health Record Incentive Program- Stage 2, also known as meaningful use of health information technology (MU2). Implementation guidance available from the Centers for Medicare and Medicaid Services (CMS) website, in conjunction with the documentation of the Quality Data Model (QDM) available from the National Quality Forum (NQF), provides high-level insights into evaluation of QDM logic that incorporates the specific occurrence construct.

Cypress is the certification testing tool for evaluating the accuracy of clinical quality measure calculations in electronic health record (EHR) systems. Specifically, the Office of the National Coordinator for Health Information Technology (ONC) designated Cypress as the official testing tool for clinical quality for the 2014 Electronic Health Record (EHR) Certification program for MU2 eCQMs. Cypress has implemented a general implementation of specific occurrences that handles the logic found in the MU2 eCQMs, as well as any well-formed logic that can be produced by the current version of the CMS Measure Authoring Tool. Cypress evaluates the MU2 eCQMs using this general framework. This document describes the approach taken within Cypress to consistent evaluation of specific occurrences.

2 Implementing Computation of Specific Occurrences

This document provides detailed guidance for software developers regarding implementation of specific occurrence evaluation in EHR technologies. This implementation guidance covers the *general implementation* for specific occurrences, which can be contrasted with the *case-by-case implementation*. The general implementation of specific occurrences is based on the constructs as defined by the underlying QDM infrastructure. The case-by-case implementation would be a translation of the QDM operators into logic that calculates a particular block of logic in an equivalent manner. For instance section 4.1 illustrates the use of specific occurrences to provide an "overlap" comparison. Implementing an "overlap" method directly rather than relying on specific occurrences for the calculation is a significantly simpler approach. However, the general solution, although complex, is required for automated consumption of eMeasure specifications. The source code for the general implementation of specific occurrences described below is available at:

https://github.com/pophealth/hqmf2js/blob/master/app/assets/javascripts/specifics.js.coffee

The generalized implementation of specific occurrences is based on the core concept that specific occurrences represent a single instance of an event, and all the logic of the measure must hold for that single event. Furthermore measures can reference multiple specific occurrences of the same type or of different types.

Based on this definition of specific occurrences, a direct method of exhaustion for implementation of specific occurrences within the CQMs would appear as the following:

```
Foreach occurrence_a_of_x in patient.getX()
        Foreach occurrence_b_of_x in patient.getX()
                Foreach occurrence_a_of_y in patient.getY()
                        …
                        Execute measure logic using each instance
                        …
```

This approach is a good starting point for understanding the final implementation of specific occurrences, but is extremely inefficient and does not provide a reliable calculation (discussed in the next paragraph). However, it is a simplified illustration of what specific occurrences are trying to calculate. The above logic would nest the iteration of each specific occurrence referenced by a measure and would evaluate the measure against every combination of specific occurrences. Each execution of the measure logic in the innermost loop would have a single event tied to each specific occurrence. In effect a "tuple" would be created by the innermost iteration. That tuple when evaluated against the measure logic would produce a true or false value. A true value would indicate that the specific combination of event instances tied to the various specific occurrences passed the measure logic. Collecting the passing tuples would provide the joint combinations of specific occurrences that evaluate to true for that measure and patient.

The issue with this simplified approach to calculation is the use of subset operators with specific occurrences. This approach resolves each specific occurrence referenced by the measure before the inception of calculation. Unfortunately this does not work well when calculating subset operators against specific occurrences. If a measure is looking for the first specific occurrence during the measurement period, or the most recent specific occurrence that start before another event, resolving the specific occurrences ahead of time to a single instance makes evaluating the set operators (first, most recent, etc.) difficult. Therefore, the nested iteration of each specific occurrence, does not work well.

Instead, the approach taken by Cypress is to evaluate specific occurrence clauses to produce a set of candidate values. These candidate values can then be intersected across logical AND statements and unioned across OR statements.

This approach requires maintaining a "Specific Context". The specific context is generated as the result of a QDM statement (e.g., Occurrence A of Encounter X during the Measurement Period). The specific context is a table with a column for each specific occurrence referenced within the measure, and each row represents a tuple of specific occurrence instances that evaluate to true at a given point of evaluation. When a QDM statement references multiple specific occurrences (e.g., Occurrence A of X starts before the start of Occurrence A of Y) the tuples in the specific context represent the combination of specific occurrence instances that evaluate to true for the logic. However, in addition to the referenced specific occurrences, the specific context tuples also have columns for specific occurrences that are not referenced in the statement. This is required because a tuple in a specific context represents the viable values across all specific occurrences, whether the statement has referenced them or not. When a specific occurrence is not referenced in a statement, all values are viable for the unreferenced specific occurrence. In other words, the result is the Cartesian product between the rows resulting from the specific occurrences referenced by the statement and all values of the unreferenced specific occurrences. This implementation uses an ANY indicator in order to represent that all values for an unreferenced specific occurrence are viable. The ANY indicator used is the '*' character and it both helps to eliminate the need to calculate the Cartesian product, but it is also useful in the calculation of negated logical statements (described below).

Therefore, every QDM statement has an associated specific context. These specific contexts are generated independently from each other (i.e., a logical statement does not depend on the result of another logical statement for calculation). The results of each QDM statement are evaluated logically through the use of AND and OR statements. As the AND and OR statements are evaluated the specific contexts generated from each statement are intersected across ANDs and unioned across ORs as the calculation progresses up the logical tree.

The process for intersecting specific contexts is to compare each tuple from the first specific context against all the tuples in the second context. Tuple comparisons are conducted by comparing each column (or specific occurrence instance) in the first tuple with the corresponding column in the second tuple. If each column refers to the same specific occurrence instance, or if one of the columns contains an ANY indicator, the tuples are equal and are added to the result

context. If one of the columns contains an ANY indicator, the intersected tuple should contain the reference to the specific occurrence instance rather than the ANY indicator.

The process for unioning specific contexts is simply to add all the tuples from each of the specific contexts being unioned to the result specific context.

Negated statements are calculated by first calculating the specific context for the positive statement. Once the specific context has been calculated, a specific context containing the Cartesian product across all values of all specific occurrences is calculated. Then to negate the positive specific context it is subtracted from the Cartesian product specific context. This calculation results in a specific context containing all the tuples that do not hold for the positive statement, thus the negation of the statement.

The one important exception to the negation logic is when the result of the positive logic returns no tuples. It seems that in this case we should subtract the empty set from the Cartesian product, which would indicate that every combination of specific occurrences is valid. While this is true we also need to account for the case where a negated clause contains references to specific occurrences that are not referenced in positive statements. If the patient record does not contain any entries that align with a specific occurrence that is only negated, the result of the Cartesian product would be the empty set. This would cause us to subtract the empty set from the empty set, which would result in the empty set. This is not the desired result. Instead, if the specific context resulting from the positive evaluation of a negated statement contains no rows, that context should be negated to a context that contains one tuple with the ANY indicator specified for each column of that tuple across all specific occurrences.

The benefit to this approach is that it allows specific occurrences that only exist in negated statements to evaluate properly. Consider the following example:

```
AND: "Occurrence A of Procedure, Performed: X" during "The measurement
Period"
AND NOT: "Occurrence B of Procedure, Performed: X" <= 60 days SBS of
"Occurrence A of Encounter, Performed: X"
```

Occurrence A of the encounter is referenced with positive statements within the measure logic. Occurrence B of the encounter is only referenced in an AND NOT statement and requires that there is not an Occurrence B of the encounter within 60 days of Occurrence A. For this logic, we want to accept patients that have just a single encounter during the measurement period. If we try to calculate the Cartesian product across Occurrence A and Occurrence B of the encounter for a patient with a single encounter, we will get the empty set for the Cartesian product. In other words, there is no viable combination of Occurrence A and Occurrence B for the patient record. Since the measure logic is verifying that Occurrence B does not exist, not having a viable Occurrence B needs to evaluate to true. Thus if we negate the empty set to a tuple containing the ANY indicator for Occurrence A and Occurrence B, this will resolve properly when this tuple is intersected with other tuples as the calculation proceeds. If the positive evaluation of a statement results in the empty set, it is a viable approach to always simply negate that result to a tuple

containing the ANY indicator for each column rather than calculating the Cartesian product and doing a subtraction.

The following example illustrates an execution of logic using this approach. In this example the logic is trying to find two heart rate physical exam findings on a patient. Both of those findings need to have been during an Office Visit and have to have a result < 50 bpm. Additionally, Occurrence B of the finding has to be the most recent finding that started before the start of Occurrence A of the finding. In other words, Occurrence B has to be immediately before Occurrence A.

```
OR:
    AND: "Occurrence A of Physical Exam, Finding: Heart Rate (result < 50
    bpm)" during "Occurrence A of Encounter, Performed: Office Visit"
    AND: "Occurrence B of Physical Exam, Finding: Heart Rate (result < 50
    bpm)" during "Occurrence A of Encounter, Performed: Office Visit"
    AND: MOST RECENT: "Occurrence B of Physical Exam, Finding: Heart Rate" SBS
    "Occurrence A of Physical Exam, Finding: Heart Rate"
```

Lets begin by labeling the three and statements AND 1 through AND 3. We define the logic of these three AND statements as:

AND 1: calculates the Occurrence A HR with result < 50bpm that are during the Occurrence A of the Encounter
AND 2: calculates the Occurrence B HR with result < 50bpm that are during the Occurrence A of the Encounter
AND 3: calculates the most recent Occurrence B HR that start before the start of Occurrence A HR

As mentioned above, we will calculate each of these three AND statements independently. Each AND statement will produce a specific context. That specific context will contain a set of tuples containing the viable set of specific occurrence instances that cause the statement to evaluate to true. Once we have calculated the specific context for the three statements, we will then intersect the three specific contexts in order to finish the calculation of this logical block.

In order to make this example more concrete lets identify specific occurrence events using simple integers. We will assume those integers represent the unique ID for the events. The events are numbered in order of occurrence, that is larger identifiers occurred later than smaller identifiers on the timeline.

Lets imagine a patient that has 5 heart rate physical exam findings. Lets also assume for simplicity that all findings have results < 50bpm (results > 50 will be covered later):

Heart Rate Findings:
1,3,5,7,8

Lets also assume that the patient has a single Encounter Performed, Office Visit with the unique

id ‘99’.

Office Visit:
99

We want to begin by calculating AND 1 and AND 2. The result of the calculation will be represented as a two dimensional array with the following structure. Each sub array will represent a single tuple. The positions of the sub array are defined as:
Position 1: Occurrence A of Heart Rate Finding
Position 2: Occurrence B of Heart Rate Finding
Position 3: Occurrence A of the Office Visit

The structure will be written as:

[[<OccurrenceA_HR>,<OccurrenceB_HR>,<OccurrenceA_OV>],[…],[…],…]

We begin by calculating AND 1 and AND 2 and get the result:

AND 1:
[[8,*,99],[7,* ,99],[5,* ,99],[3,* ,99],[1,* ,99]]
AND 2:
[[*,8,99],[*,7,99],[*,5,99],[*,3,99],[*,1,99]]

The assumptions that we made earlier mean that all five heart rate readings align with the logic of AND 1 and AND 2. As a result, we have a tuple in the resulting specific context for each of these values. For AND 1 the heart rate entries are in position 1 and in AND 2 the entries are in position 2. Note that the ANY indicator (‘*’), is used in position 2 of AND 1 and position 1 of AND 2. The reason for this is that these positions represent unreferenced specific occurrences. AND 1 references Occurrence A but not B while AND 2 references Occurrence B but not A. The ANY indicator represents that ANY value of that occurrence is valid.

Next we need to calculate AND 3. We begin by calculating the temporal reference and then apply the subset operator (see order of operations).

The temporal reference for AND 3 is
(Occurrence B HR that start before the start of Occurrence A HR)

The result of this calculation is:
[[8,7,*],[8,5,*],[8,3,*],[8,1,*],[7,5,*],[7,3,*],[7,1,*],[5,3,*],[5,1,*],[3,1,*]]

Note the ANY indication in position 3 (since the encounter is not referenced). Also note that for each sub array above, position 2 is less than position 1. Since we are looking for Occurrence B to start before the start of Occurrence A and position 2 represents Occurrence B and position 1 represents Occurrence A, this result represents the specific occurrence instances that hold for the logic.

Once the temporal reference has been calculated, the subset operator ("most recent") must be applied. Since the subset is after temporal references in the order of operations the subset operator is applied after the calculation of the temporal reference. We are calculating the subset on Occurrence B since it is on the left side of the evaluation in AND 3. In order to do this calculation, we must "group" the occurrence B entries by all other entries in the tuple (in this case Occurrence A). The reason we group the specific occurrences is that we are evaluating the subset operator on Occurrence B given an Occurrence A value. The subset operator cannot be calculated globally on the tuples since the set of values for Occurrence B is dependent on a specific Occurrence A value. If the subset were calculated globally, it would be restrict the viable set of Occurrence A values based on a calculation against Occurrence B without considering the dependence on Occurrence A. By grouping Occurrence B by all other specific occurrences and then applying the subset operator to each grouping of B, we are selecting the most recent B given A.

Consider the following example:

```
AND: First: "Occurrence A of Physical Exam, Finding: Blood Pressure"
during "Occurrence A of Encounter, Performed: Office Visit"
```

In this case we want to find the first blood pressure reading that occurred during an office visit. We cannot apply the First operator across all of the patient's blood pressure readings. Instead, we need to find the first blood pressure reading for each encounter. Thus, if we group the blood pressure readings by encounter, then select the first from each of those sets, we get the correct tuples. Remember that we will most likely be intersecting the results of this evaluation with the results of other statements. Therefore, if we have a patient with multiple encounters that had blood pressure readings, some of those encounters could be eliminated by additional logic. Thus we must maintain the first blood pressure reading with its associated encounter across all viable encounters. The tuples maintained by the specific occurrence allow us to do this. In this case, as calculation progresses the logic may exclude either the physical exam finding or the encounter contained in the tuples returned by this statement. Since the tuples "join" a given instance of the physical exam finding with an instance of the encounter (identifying the finding as the first for that encounter), excluding one will exclude the other.

Therefore, in our original example if we group the calculation of the "most recent" subset operator we get the following set of groups:

Occurrence A=8: [8,7,*],[8,5,*],[8,3,*],[8,1,*]
Occurrence A=7: [7,5,*],[7,3,*],[7,1,*]
Occurrence A=5: [5,3,*],[5,1,*]
Occurrence A=3: [3,1,*]

Note that for each unique value of Occurrence A we have a group defined. For each of these groups we apply the subset operator (in this case "most recent") to the Occurrence B value in the group. Consider the group where Occurrence A equals 8. As defined previously the integers representing events also signify the timing of the events. Therefore the tuple where Occurrence

B equals 7 is the latest, or most recent. Thus we eliminate the remaining three tuples where Occurrence B equals 5, 3 and 1. The result of the calculation of the subset operator for each group is the final result of AND 3 which is:

[[8,7,*],[7,5,*],[5,3,*],[3,1,*]]

We now need to intersect the results of AND 1,2 and 3. The results we have calculated for each statement to this point are the following:

AND 1:
[[8,*,99],[7,* ,99],[5,* ,99],[3,* ,99],[1,* ,99]]
AND 2:
[[*,8,99],[*,7,99],[*,5,99],[*,3,99],[*,1,99]]
AND 3:
[[8,7,*],[7,5,*],[5,3,*],[3,1,*]]

We begin by calculating the intersection of AND 1 and AND 2. The result is the cross product of the values with the rows where OccurrenceA == OccurrenceB are removed, I.e.,

AND 1 INTESECT AND 2:
[[8,7,99],[8,5,99],[8,3,99],[8,1,99],[7,8,99],[7,5,99],[7,3,99],
[7,1,99], … [1,8,99],[1,7,99],[1,5,99],[1,3,99]]

We now intersect the result of AND 3 with the intersection of AND1/AND2. Since AND3 is a proper subset of AND1/AND2 with respect to the Heart Rate findings we get the result of AND3 returned from the intersection. The difference in the result is that the ANY indicators for the encounter in the original AND 3 result have now been resolved as part of the intersection. The result of the intersection is:

[[8,7,99],[7,5,99],[5,3,99],[3,1,99]]

What this result actually means is that we have 4 viable combinations of these three specific occurrences that hold for the measure logic. If this logic were part of a larger measure this result would be intersected and unioned as calculation progressed though the evaluation of logical AND and OR statements through the rest of the measure. Once the calculation reaches the logical root of a measure population (i.e., the parent most AND or OR statement of the Denominator, or Numerator), an evaluation needs to be done to verify that rows exist in the specific context. If at least one row exists in the context then the specific occurrence logic holds for that population.

Next lets remove the assumption that all heart rate values have values <50bpm. Assume that entry 5 had a result of 75 bpm. The tuples for entry 5 would be removed from the results of AND 1 and AND2. As a result, the intersection between AND3 and AND1/AND2 would remove the tuples that contain entry 5 since that entry would not exist in the intersection of AND1/AND2. The result would therefore be.

[[8,7,99],[3,1,99]]
Similarly, the assumption that we have just one encounter could also be handled. If we had multiple encounters those values would be considered as part of the logic and intersections in order to complete the calculation.

Using this approach to calculating specific occurrences it is possible to discretely calculate specific occurrences using the QDM.